To : Members of the COMAL Standardisation and Development Groups
From : David Christensen
Ian Malcolm
Ref : **EXTENSIONS TO COMAL**
Date : 21 August 1985

Introduction

Since the last meeting of the Standardisation and Development Groups we have been considering the implementation of extensions to COMAL. As a result, we wish to bring forward the attached Proposals. Because these proposals represent a slight departure from recent thinking in this area, it is pertinent to indicate, briefly, some of the thinking behind the Proposals.

Background

We seem to be agreed, at least informally, that packages, written in machine code, should provide a flexible method of extending the language. This "flexible method of extending the language" is a key point and it seems sensible that the user should be able to extend the language, for his own use, in directions relevant to him. Because the average user cannot produce machine code packages for himself, the idea of libraries has grown up. Libraries, written in COMAL, provide the user with such a method of extending the language.

Although these are two distinct methods of extension they should, when in use, be indistinguishable to the user. This means that any commands for handling extensions must be able to cope with both types. Essentially the facilities we need are to be able to:

1. allow the addition and removal of extensions (libraries / packages) at run time;

2. allow the use of more that one extension at any time; and

3. provide a method of overcoming name clashes.

The attached Proposals follow from this argument and provide a method of implementing the facilities noted.

PROPOSAL 1

EXTENSIONS

Need

Following the last two meetings of the Standardisation and Development Groups the need for mechanisms to extend the language has been demonstrated. This proposal relates to this need. In particular, it heeds the suggestions from the last meeting that libraries and packages should be grouped into a single concept, thus making life easier for the user.

Proposal

Production 6 should be modified (as the Extension Standard Extension) to include the following clauses:

a. <include statement> ::=
INCLUDE <filename><eol>

b. <exclude statement> ::=
EXCLUDE <filename><eol>

c. <use statement> ::=
USE <use element> **FROM** <filename><eol>

d. <use element> ::=
<function identifier> | <procedure identifier>

Semantics

In the following description "procedure" should be interpreted as "function or procedure".

Extension files may be either COMAL program or machine code files.

The extension statements (a, b, c and d) may be used at command level or in programs at the level of the main program (i.e. they may not be used inside procedures).

The effect of the statements is dynamic.

When an **INCLUDE** statement is encountered the system searches for the specified file and ensures that it is of a correct format. If it is not found or is not valid a run time error occurs. Otherwise the procedures in the file are added to the list of those available in the kernel of the language. (i.e. They are to be considered to have global scope and will not have to be **IMPORT**ed into **CLOSED** contexts.) At the system's discretion the extension file may be loaded into the system's primary memory on declaration or when required for use; similarly, it may be removed from primary memory at any time when not in use.

When a procedure call is made the program will be searched for that procedure (following the normal rules of scope). If it is not found the currently **INCLUDE**d extensions are searched in the order of their declaration (this is more natural than searching in the reverse of the declared order - see also Proposal 2). The first occurrence of the required name will be located and used. This may be modified by using the **USE** statement: the **USE** statement will cause the system to ignore all occurrences of a particular procedure name except that occurring in the specified extension file.

When no longer required an extension may be **EXCLUDE**d. All extensions will be implicitly **EXCLUDE**d when a **NEW** or **RUN** is executed or when a new program is loaded into memory.

PROPOSAL 2

PRECEDENCE OF PROCEDURES AND FUNCTIONS

Need

If two or more procedures (or functions) with the same name exist in the current scope, there must be some way of deciding which should be executed in response to a procedure (or function) call. This proposal deals with the order in which the current scope is searched.

Proposal

That the following be added to the notes after Production 135:

"The procedure executed in response to to a procedure call should be the first procedure found which has the same <procedure identifier> as is in the call when the current scope is searched in the order:

- reserved words in COMAL;
- identifiers in current **USE**;
- user's program in line number order;
- currently **INCLUDE**d extensions in order of their declaration."

That the following be added after Production 86:

"The function executed in response to an integer function call should be the first function found which has the same <function identifier> as is in the call when the current scope is searched in the order:

- reserved words in COMAL;
- identifiers in current **USE**;
- user's program in line number order;
- currently **INCLUDE**d extensions in order of their declaration."

That the following be added after Production 95:

"The function executed in response to to a string function call should be the first function found which has the same <function identifier> as is in the call when the current scope is searched in the order:

- reserved words in COMAL;
- identifiers in current **USE**;
- user's program in line number order;
- currently **INCLUDE**d extensions in order of their declaration."

PROPOSAL 3

GLOBAL

Need
If a procedure or function is required within several other procedures or functions it may be inconvenient to have to **IMPORT** it into each of the required contexts. This may be solved if we allow a definition to be qualified as being **GLOBAL.**

Proposal
That Production 39 be modified to read:

39. <head appendix> ::=
[(<formal parameter list>)] [**CLOSED**] [**GLOBAL**]

Semantics (add to para 3 following Production 46)
The qualifier **GLOBAL** will make the procedure or function always available, even within **CLOSED** contexts, without having to explicitly **IMPORT** it.